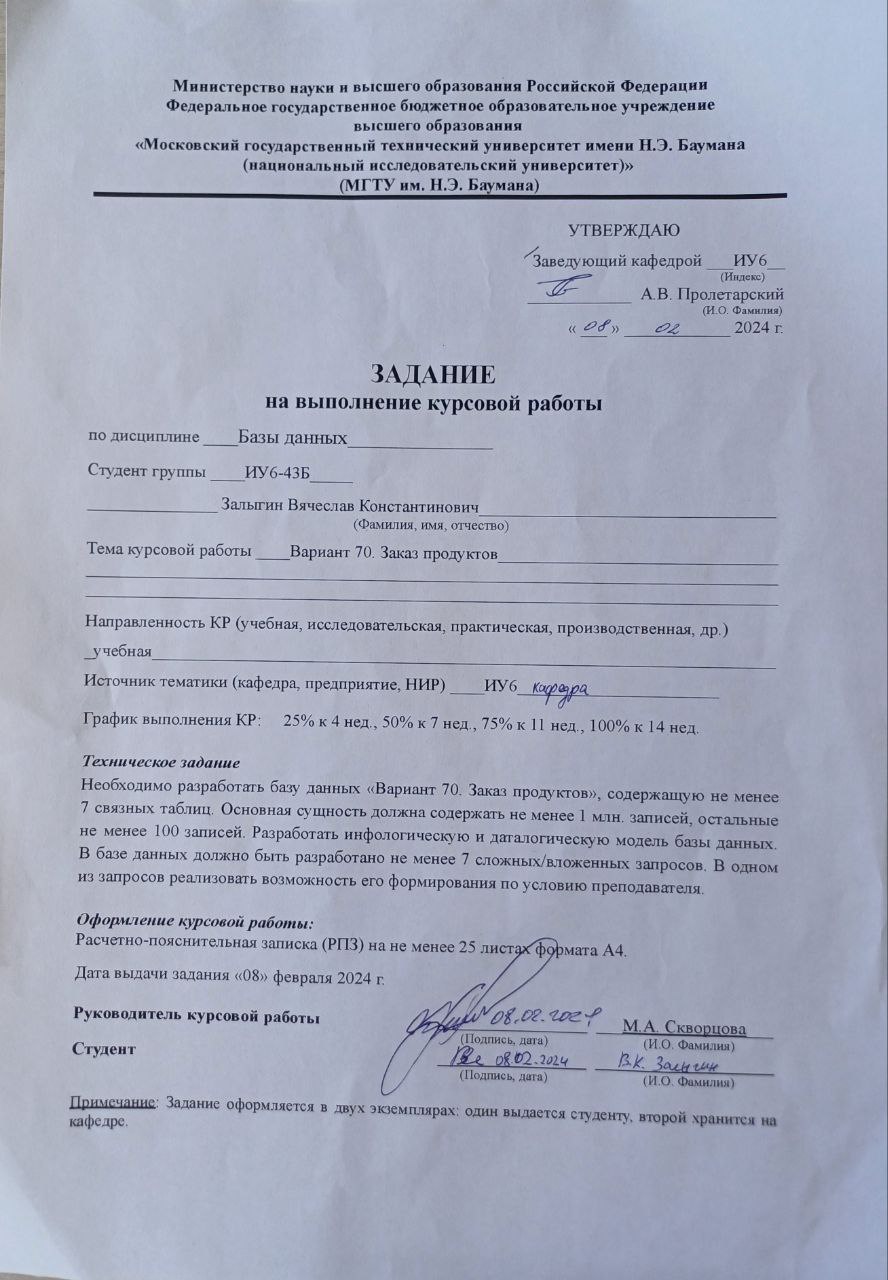
# 

****

# **СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 2](#_Toc166336492)

[ВВЕДЕНИЕ 3](#_Toc166336493)

[1 ПРОЕКТИРОВАНИЕ 5](#_Toc166336494)

[1.1 Анализ предметной области 5](#_Toc166336495)

[1.2 Выделение сущностей 6](#_Toc166336496)

[1.3 Проектирование инфологической модели базы данных. 7](#_Toc166336497)

[1.4 Проектирование даталогической модели базы данных. 8](#_Toc166336498)

[2 РЕАЛИЗАЦИЯ 14](#_Toc166336499)

[2.1 Написание скрипта создания базы данных 14](#_Toc166336500)

[2.2 Заполнение базы данных 28](#_Toc166336501)

[2.3 Сложные запросы 36](#_Toc166336502)

[ЗАКЛЮЧЕНИЕ 41](#_Toc166336503)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 42](#_Toc166336504)

[ПРИЛОЖЕНИЕ А. 43](#_Toc166336505)

# **ВВЕДЕНИЕ**

В последние годы активно развиваются сети продуктовых магазинов, в основе которых лежит модель дарксторов (“магазины без покупателей”) с мгновенной доставкой продуктов. Такая модель становится удобна конечному потребителю, так как позволяет в течение короткого промежутка времени получить необходимые продукты и не тратить время на самостоятельный их закуп. Так как бизнес-модель основана на сверхбыстрой доставке (не более 30 минут с момента оформления заказа), предприятия ее реализующие у предъявляют высокие требования к уровню организованности и эффективности оконечных точек сети доставки – дарксторов. Создаваемая база данных предназначена для обеспечения эффективной работы сети дарксторов с мгновенной доставкой.

Актуальность данной системы заключается в возможности эффективной организации операционной деятельности оконечных точек сетей с вышеописанной бизнес-моделью. Система позволяет контролировать ассортимент продуктов, состояние заказов, доставок, поставок со склада, смены персонала оконечной точки, а также проводить анализ операционной деятельности с целью подсчета бизнес-метрик и повышения эффективности предприятия. Данная система позволяет повысить качество и скорость обслуживания пользователей, что является важнейшими показателями для бизнеса.

Целью работы является создание системы, которая обеспечит эффективную операционную деятельностью оконечных точек сети дарксторов.

Для достижения поставленной цели необходимо решить следующие задачи:

* Проведение анализа предметной области заказа продуктов, выделение основных сущностей и процессов, определение требований к базе данных;
* Проектирование базы данных, разработка инфологической и даталогической модели базы данных;
* Написание скрипта для создания базы данных со всеми инструментами обеспечения консистентности хранящейся информации;
* SQL-запросы для получения статистической информации.

Для реализации используется реляционная база данных PostgerSQL 16, графический клиент DBeaver 23.2.3, язык Python 3.10, среда разработки Visual Studio Code.

# **1 ПРОЕКТИРОВАНИЕ**

## **1.1 Анализ предметной области**

В сфере заказа продуктов существуют различные бизнес-процессы, поддерживающие возможность выбора продуктов и их доставки до клиента. Ниже рассмотрены основные аспекты данной предметной области.

Предметной областью данного проекта является управление операционной деятельностью предприятия, позволяющего клиенту осуществлять заказы продуктов. База данных предоставляет возможность организации заказов, сборки и доставки списка продуктов на основе актуального ассортимента, учета продуктов и их своевременного списания по достижению срока хранения, рабочего цикла на предприятиях и распределения людей на роли.

Функционал базы данных включает в себя:

* Запись и учет данных о поставках партий продуктов в оконечные точки, включая количество поставляемых продуктов и сроки хранения партии;
* Запись и учет данных о клиенте, включая имя, фамилию, контактную информацию, сохранённые адреса;
* Хранение и формирование, обработка информации о заказах пользователя, состояниях заказа, доставки и сборки заказанных продуктов;
* Запись и учет информации о сотрудниках и выставляемых сменах;
* Запись и учет данных об ассортименте дарксторов на основе прихода (поставок) и ухода (заказов);
* Списание партий продуктов с истекшим сроком хранения;
* Создание заданий для сборщиков и доставщиков на основе состояний о текущих заказах.

Данная база данных является актуальной и эффективной, поскольку позволяет управлять информацией о заказах, поставках и сборках, обеспечивая консистентность данных и быстрый доступ к необходимой информации для управления процессом осуществления заказа продуктов.

## **1.2 Выделение сущностей**

Для проектирования базы данных необходимо выделить элементы предметной области. В рамках анализа можно выделить следующие сущности:

1. Клиент (Client):

Данная сущность содержит информацию о клиентах предприятия. Атрибуты: уникальный номер, персональные данные клиентов, такие как имя, фамилия, сохранённые адреса доставки, контактный телефон, почта.

1. Заказ (Order):

Данная сущность хранит информацию о заказе продуктов. Атрибуты: уникальный номер, состояние заказа, клиент, сделавших заказ, сборки продуктов, адрес доставки, время создания, время начала доставки, время закрытия заказа, дарстор, из которого осуществляется доставка.

1. Продукт (Product):

Хранит информацию о продукте, продающемся предприятием. Атрибуты: уникальный номер, название, описание, изображение продукта, цена за упаковку.

1. Сборка

Хранит информацию о сборке продуктов для заказов. Атрибуты: продукт, заказ, количество продуктов, время создания начала сборки, время окончания сборки, статус.

1. Доставка

Хранит информацию о доставках заказов. Атрибуты: уникальный номер, доставляемый заказ, доставляющий курьер, время создания, статус доставки.

1. Сотрудник

Хранит информацию о сотруднике, работающем в сети. Атрибуты: уникальный номер, имя, фамилия, должность.

1. Смена

Хранит информацию о смене сотрудников дарксторов. Атрибуты: уникальный номер, сотрудник, время начала и окончания смены, магазин, в котором проходит смена.

1. Поставка

Хранит информацию о поставках партий продуктов в дарксторы. Атрибуты: уникальный номер, магазин, продукт, время доставки, окончание срока годности партии, статус поставки, количество единиц продукта.

1. Ассортимент

Хранит информацию об остатках продуктов в определенном дарксторе. Атрибуты: продукт, магазин, остаток (количество).

1. Даркстор

Хранит информацию об оконечной точке сети. Атрибуты: уникальный номер, адрес, техническое имя, владелец.

1. Расположение продукта

Хранит информацию о расположении продукта в рамках одного даркстора для обеспечения быстрого поиска продуктов и сборки заказа. Атрибуты: продукт, магазин, описание расположения (буквенно-численный код, обозначающий местоположение продукта)

Главной сущностью в системе является Заказ.

## **1.3 Проектирование инфологической модели базы данных**

Исходя из сущностей и их свойств, определенных в пункте 1.2, возможно построить инфологическую модель базы данных. Это необходимо для дальнейшего корректного проектирования базы данных. На рисунке 1 представлена инфологическая модель БД.

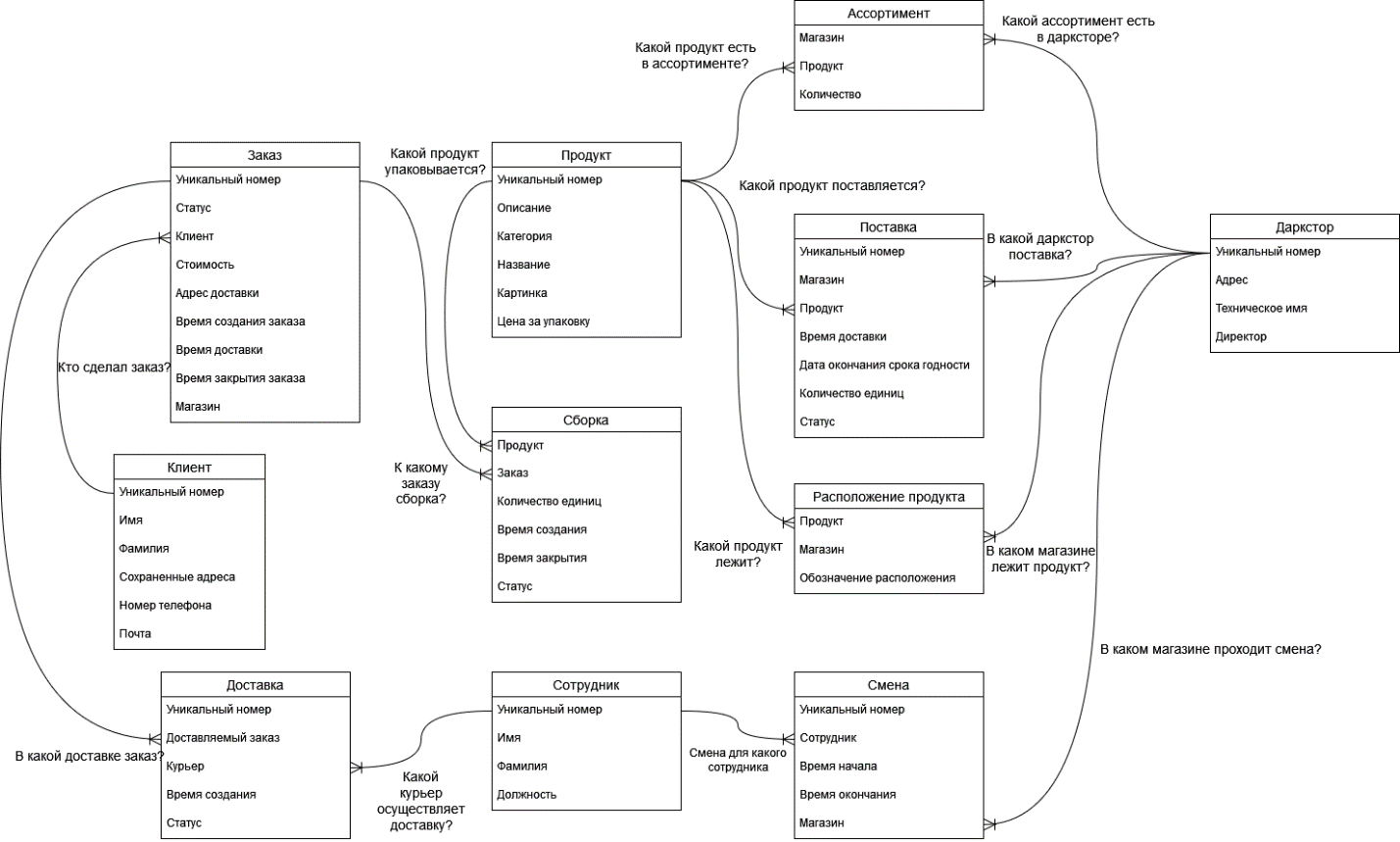


Рисунок 1 – Инфологическая модель БД

При разработке инфологической модели были выделены основные процессы, через которые происходит взаимодействие сущностей.

## **1.4 Проектирование даталогической модели базы данных**

На основании инфологической модели можно построить даталогическую модель, необходимую для написания корректного скрипта создания базы данных и входящих в нее таблиц.

Из пунктов 1.3 и 1.4 следуют следующие основные функции, которые должна реализовывать разрабатываемая база данных:

1) Управление и хранение данных о клиентах;

2) Управление и хранение данных о продуктах, ассортименте и сопутствующих сущностях;

3) Управление и хранение данных о сотрудниках;

4) Управление и хранение необходимой дополнительной информации;

5) Поддержание консистентного состояния данных на основе триггеров.

На основании инфологической модели, и выделенных из нее функций, была построена даталогическая модель, представленная на рисунке 2.

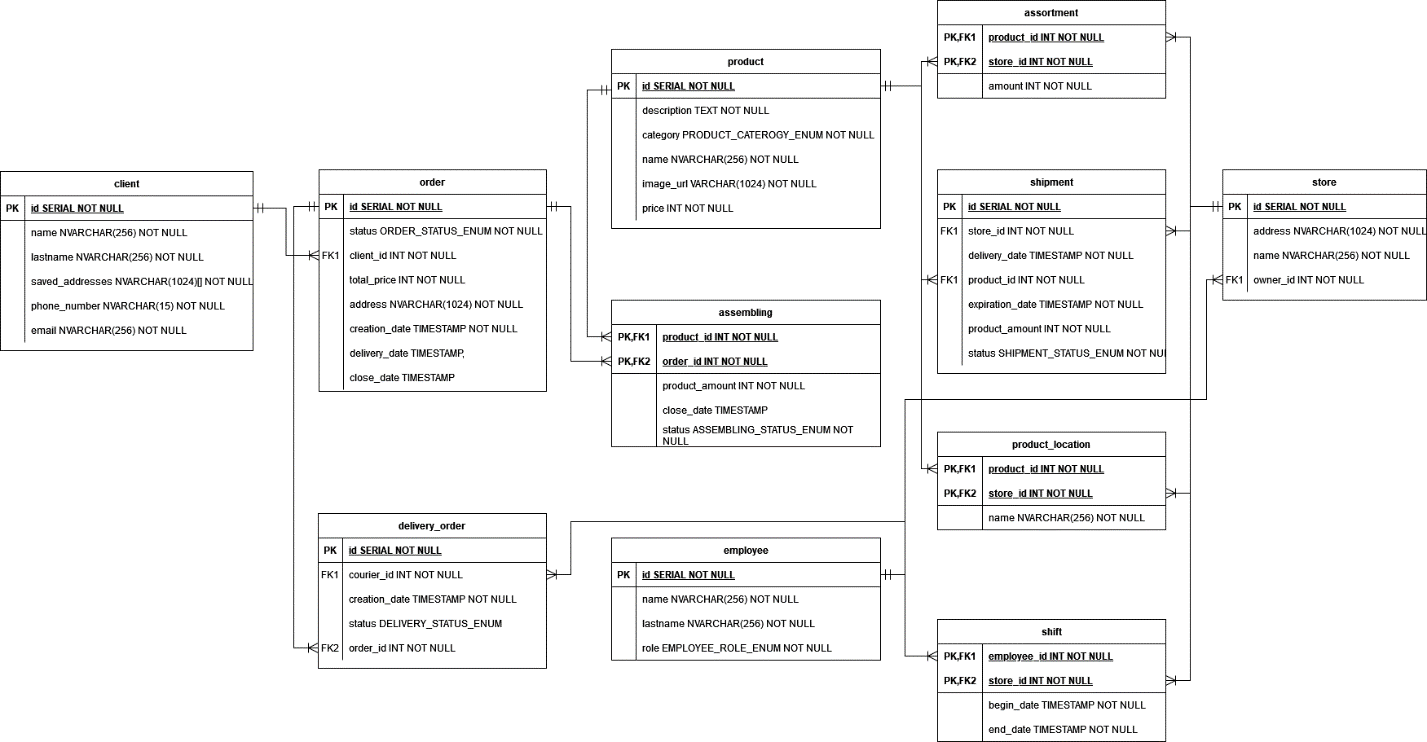


Рисунок 2 – Даталогическая модель БД

Для того, чтобы пояснить назначение полей в каждой из таблиц, а связи между ними были более понятными, ниже приведено описание каждой из таблиц и ее полей.

Таблица 1 – Описание таблицы “clients”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор | Целочисленный последовательный |
| name | Фамилия | Строковый |
| lastname | Имя | Строковый |
| saved\_addresses | Сохранённые адреса | Массив строковый |
| phone\_number | Телефон | Строковый |
| email | Почта | Строковый |

Таблица 2 – Описание таблицы “order”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор каждого заказа | Целочисленный последовательный |
| status | Статус | Перечисление статусов |
| client\_id | Идентификатор клиента | Целочисленный |
| total\_price | Общая стоимость заказа | Целочисленный |
| address | Адрес доставки для заказа | Строковый |
| creation\_date | Дата и время создания заказа | Дата и время |
| delivery\_date | Дата и время готовности заказа для доставки | Дата и время |
| close\_date | Дата и время закрытия заказа | Дата и время |
| store\_id | Идентификатор магазина, где был размещен заказ | Целочисленный |

Таблица 3 – Описание таблицы “assembling”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор каждой доставки | Целочисленный последовательный |
| order\_id | Идентификатор связанного заказа | Целочисленный |
| courier\_id | Идентификатор курьера, выполняющего доставку | Целочисленный |
| status | Статус доставки | Перечисление статусов |

Таблица 4 – Описание таблицы “delivery”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор каждого сотрудника | Целочисленный последовательный |
| name | Имя сотрудника | Строковый |
| lastname | Фамилия сотрудника | Строковый |
| role | Роль сотрудника | Перечисление статусов |

Таблица 5 – Описание таблицы “employee”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор каждого сотрудника | Целочисленный последовательный |
| name | Имя сотрудника | Строковый |
| lastname | Фамилия сотрудника | Строковый |
| role | Роль сотрудника | Перечисление статусов |

Таблица 6 – Описание таблицы “shift”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| employee\_id | Идентификатор сотрудника, работающего сменой | Целочисленный |
| store\_id | Идентификатор магазина, где работает сотрудник | Целочисленный |
| begin\_date | Дата и время начала смены | Дата и время |
| end\_date | Дата и время конца смены | Дата и время |

Таблица 7 – Описание таблицы “product\_location”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| product\_id | Идентификатор продукта | Целочисленный |
| store\_id | Идентификатор магазина | Целочисленный |
| description | Описание расположения продукта | Строковый |

Таблица 8 – Описание таблицы “product”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор продукта | Целочисленный последовательный |
| name | Название продукта | Строковый |
| description | Описание продукта | Строковый |
| category | Категория продукта | Перечисление статусов |
| image\_url | URL-адрес изображения продукта | Строковый |
| price | Цена продукта | Целочисленный |

Таблица 9 – Описание таблицы “assortment”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| store\_id | Идентификатор магазина | Целочисленный |
| product\_id | Идентификатор продукта | Целочисленный |
| amount | Количество продукта в ассортименте определенного магазина | Целочисленный |

Таблица 10 – Описание таблицы “shipment”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор поставки | Целочисленный последовательный |
| store\_id | Идентификатор магазина, куда поставляется товар | Целочисленный |
| product\_id | Идентификатор продукта в поставке | Целочисленный |
| delivery\_date | Дата и время доставки поставки | Дата и время |
| expiration\_date | Дата и время истечения срока годности партии продуктов | Дата и время |
| product\_amount | Количество продуктов в поставке | Целочисленный |
| status | Статус поставки | Перечисление статусов |

Таблица 11 – Описание таблицы “store”

|  |  |  |
| --- | --- | --- |
| Название поля | Предназначение поля | Тип данных |
| id | Уникальный идентификатор магазина | Целочисленный последовательный |
| address | Адрес магазина | Строковый |
| name | Название магазина | Строковый |
| owner\_id | Идентификатор владельца магазина | Целочисленный |

# **2 Реализация**

## **2.1 Написание скрипта создания базы данных**

При задании схемы базы данных первоочередно создаются типы данных, используемых в таблицах. В листинге 1 представлен скрипт, создающий необходимые типы-перечисления для таблиц.

Листинг 1 – Создание перечислений

|  |
| --- |
| DROP TYPE IF EXISTS ORDER\_STATUS\_ENUM;  CREATE TYPE ORDER\_STATUS\_ENUM AS ENUM (  'assembling', 'assembled', 'delivering', 'closed', 'cancelled'  );  DROP TYPE IF EXISTS DELIVERY\_STATUS\_ENUM;  CREATE TYPE DELIVERY\_STATUS\_ENUM AS ENUM (  'scheduled', 'on\_the\_way', 'closed'  );  DROP TYPE IF EXISTS EMPLOYEE\_ROLE\_ENUM;  CREATE TYPE EMPLOYEE\_ROLE\_ENUM AS ENUM (  'courier', 'assembler', 'manager'  );  DROP TYPE IF EXISTS PRODUCT\_CATEGORY\_ENUM;  CREATE TYPE PRODUCT\_CATEGORY\_ENUM AS ENUM (  'alcohol', 'bakery', 'bread', 'cheese', 'fish\_and\_seafood', 'fruit', 'juice', 'lemonade', 'meat', 'milk\_and\_egg', 'sausage', 'snack', 'water', 'yogurt'  );  DROP TYPE IF EXISTS SHIPMENT\_STATUS\_ENUM;  CREATE TYPE SHIPMENT\_STATUS\_ENUM AS ENUM (  'on\_the\_way', 'delivered', 'accepted', 'run\_out', 'expired'  ); |

Типы с окончанием STATUS\_ENUM перечисляют множество значений соответствующей сущности. Тип PRODUCT\_CATEGORY\_ENUM перечисляет категории продуктов.

Листинг 2 содержит операторы создания таблиц.

Листинг 2 – Скрипт создания таблиц

|  |
| --- |
| CREATE TABLE IF NOT EXISTS client (  id SERIAL PRIMARY KEY,  "name" VARCHAR(256) NOT NULL,  lastname VARCHAR(256) NOT NULL,  saved\_addresses VARCHAR(1024)[] NOT NULL,  phone\_number VARCHAR(256) NOT NULL,  email VARCHAR(256) NOT NULL  );  CREATE TABLE IF NOT EXISTS "order" (  id SERIAL,  "status" ORDER\_STATUS\_ENUM NOT NULL DEFAULT 'assembling',  client\_id INT NOT NULL,  total\_price INT NOT NULL CHECK (total\_price >= 0) DEFAULT 100,  "address" VARCHAR(1024) NOT NULL,  creation\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  delivery\_date TIMESTAMP,  close\_date TIMESTAMP,  store\_id INT NOT NULL,  PRIMARY KEY (id),  CHECK (delivery\_date = null or delivery\_date >= creation\_date),  CHECK (close\_date = null or delivery\_date != null and close\_date >= delivery\_date)  );  CREATE TABLE IF NOT EXISTS delivery (  id SERIAL,  order\_id INT NOT NULL,  courier\_id INT NOT NULL,  "status" DELIVERY\_STATUS\_ENUM,  PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS employee (  id SERIAL,  "name" VARCHAR(256) NOT NULL,  lastname VARCHAR(256) NOT NULL,  "role" EMPLOYEE\_ROLE\_ENUM NOT NULL,  PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS product (  id SERIAL,  "name" VARCHAR(256) NOT NULL,  "description" TEXT NOT NULL,  category PRODUCT\_CATEGORY\_ENUM NOT NULL,  image\_url VARCHAR(1024) NOT NULL,  price INT NOT NULL,  PRIMARY KEY (id),  CHECK (price >= 0)  );  CREATE TABLE IF NOT EXISTS shipment (  id SERIAL,  store\_id INT NOT NULL,  product\_id INT NOT NULL,  delivery\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  expiration\_date TIMESTAMP NOT NULL,  product\_amount INT NOT NULL,  "status" SHIPMENT\_STATUS\_ENUM NOT NULL DEFAULT 'delivered',  PRIMARY KEY (id),  CHECK (product\_amount >= 0)  );  CREATE TABLE IF NOT EXISTS assortment (  store\_id INT NOT NULL,  product\_id INT NOT NULL,  amount INT NOT NULL,  PRIMARY KEY (store\_id, product\_id),  CHECK (amount >= 0)  );  CREATE TABLE IF NOT EXISTS assembling (  product\_id INT NOT NULL,  order\_id INT NOT NULL,  product\_amount INT NOT NULL CHECK (product\_amount > 0),  creation\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  close\_date TIMESTAMP,    PRIMARY KEY (product\_id, order\_id),    CHECK (close\_date = null or close\_date > creation\_date)  );  CREATE TABLE IF NOT EXISTS store (  id SERIAL,  "address" VARCHAR(1024) NOT NULL,  "name" VARCHAR(256) NOT NULL,  "owner\_id" INT NOT NULL,  PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS product\_location (  product\_id INT NOT NULL,  store\_id INT NOT NULL,  "description" VARCHAR(256) NOT NULL,  PRIMARY KEY (product\_id, store\_id)  );  CREATE TABLE IF NOT EXISTS shift (  employee\_id INT NOT NULL,  store\_id INT NOT NULL,  begin\_date TIMESTAMP NOT NULL,  end\_date TIMESTAMP NOT NULL,  PRIMARY KEY (employee\_id, store\_id),  CHECK (end\_date > begin\_date)  ); |

После создания таблиц следует определения внешних ключей. Листинг 3 содержит скрипт задания внешних ключей в соответствии с даталогической моделью.

Листинг 3 – Создание внешних ключей

|  |
| --- |
| ALTER TABLE "order" ADD CONSTRAINT fk\_order\_client FOREIGN KEY (client\_id) REFERENCES client;  ALTER TABLE "order" ADD CONSTRAINT fk\_order\_store FOREIGN KEY (store\_id) REFERENCES store;  ALTER TABLE delivery ADD CONSTRAINT fk\_delivery\_courier FOREIGN KEY (courier\_id) REFERENCES employee;  ALTER TABLE delivery ADD CONSTRAINT fk\_delivery\_order FOREIGN KEY (order\_id) REFERENCES "order";  ALTER TABLE shipment ADD CONSTRAINT fk\_shipment\_store FOREIGN KEY (store\_id) REFERENCES store;  ALTER TABLE shipment ADD CONSTRAINT fk\_shipment\_product FOREIGN KEY (product\_id) REFERENCES product;  ALTER TABLE store ADD CONSTRAINT fk\_store\_owner FOREIGN KEY ("owner") REFERENCES employee;  ALTER TABLE assembling ADD CONSTRAINT fk\_assembling\_product FOREIGN KEY (product\_id) REFERENCES product;  ALTER TABLE assembling ADD CONSTRAINT fk\_assembling\_order FOREIGN KEY (order\_id) REFERENCES "order";  ALTER TABLE product\_location ADD CONSTRAINT fk\_product\_location\_product FOREIGN KEY (product\_id) REFERENCES product;  ALTER TABLE product\_location ADD CONSTRAINT fk\_product\_location\_store FOREIGN KEY (store\_id) REFERENCES store;  ALTER TABLE shift ADD CONSTRAINT fk\_shift\_employee FOREIGN KEY (employee\_id) REFERENCES employee;  ALTER TABLE shift ADD CONSTRAINT fk\_shift\_store FOREIGN KEY (store\_id) REFERENCES store;  ALTER TABLE assortment ADD CONSTRAINT fk\_assortment\_store FOREIGN KEY (store\_id) REFERENCES store;  ALTER TABLE assortment ADD CONSTRAINT fk\_assortment\_product FOREIGN KEY (product\_id) REFERENCES product; |

Для обеспечения согласованности данных необходимы триггеры. Листинг 4 содержит скрипт для создания триггеров.

Листинг 4 – Создание триггеров

|  |
| --- |
| CREATE OR REPLACE FUNCTION set\_order\_assembled\_on\_assembly\_ready\_func()  RETURNS TRIGGER AS  $$  BEGIN  IF  (SELECT count(\*)  FROM assembling  WHERE assembling.order\_id = NEW.order\_id AND assembling.close\_date = null)  = 0  THEN  UPDATE "order"  SET "status" = 'assembled'  WHERE "order".id = NEW.order\_id;  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER set\_order\_assembled\_on\_assembly\_ready  AFTER UPDATE  ON assembling  FOR EACH ROW  EXECUTE PROCEDURE set\_order\_assembled\_on\_assembly\_ready\_func();  CREATE OR REPLACE FUNCTION update\_assortment\_on\_shipment\_accepted\_func()  RETURNS TRIGGER AS  $$  BEGIN  IF  OLD."status" != NEW."status" AND NEW."status" = 'accepted'  THEN  IF (  SELECT count(\*) FROM assortment WHERE assortment.store\_id = NEW.store\_id AND assortment.product\_id = NEW.product\_id  ) = 0 THEN  INSERT INTO assortment (store\_id, product\_id, amount)  VALUES (NEW.store\_id, NEW.product\_id, 0);  END IF;  UPDATE assortment  SET amount = amount + NEW.product\_amount  WHERE assortment.store\_id = NEW.store\_id AND assortment.product\_id = NEW."product\_id";  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER update\_assortment\_on\_shipment\_accepted  AFTER UPDATE  ON shipment  FOR EACH ROW  EXECUTE PROCEDURE update\_assortment\_on\_shipment\_accepted\_func();  CREATE OR REPLACE FUNCTION decrease\_product\_amount\_on\_new\_assembling\_func()  RETURNS TRIGGER AS  $$  DECLARE  store\_id\_var INT := (  SELECT "order".store\_id FROM "order" WHERE "order".id = NEW.order\_id  );  BEGIN  UPDATE assortment  SET amount = amount - NEW.product\_amount  WHERE assortment.store\_id = store\_id\_var AND assortment.product\_id = NEW.product\_id;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER decrease\_product\_amount\_on\_new\_assembling  BEFORE INSERT  ON assembling  FOR EACH ROW  EXECUTE PROCEDURE decrease\_product\_amount\_on\_new\_assembling\_func();  CREATE OR REPLACE FUNCTION set\_shipment\_run\_out\_on\_zero\_amount\_func()  RETURNS TRIGGER AS  $$  BEGIN  IF  NEW.product\_amount = 0 AND NEW.status = 'accepted'  THEN  UPDATE shipment  SET "status" = 'run\_out'  WHERE shipment.id = NEW.id;  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER set\_shipment\_run\_out\_on\_zero\_amount  AFTER UPDATE  ON shipment  FOR EACH ROW  EXECUTE PROCEDURE set\_shipment\_run\_out\_on\_zero\_amount\_func();  CREATE OR REPLACE FUNCTION set\_order\_timestamp\_on\_status\_change\_func()  RETURNS TRIGGER AS  $$  BEGIN  IF  NEW."status" = 'delivering' AND OLD."status" != 'delivering'  THEN  UPDATE "order"  SET delivery\_date = CURRENT\_TIMESTAMP  WHERE "order".id = NEW.id;  END IF;  IF  (NEW."status" = 'closed' AND OLD."status" != 'closed') OR  (NEW."status" = 'cancelled' AND OLD."status" != 'cancelled')  THEN  UPDATE "order"  SET close\_date = CURRENT\_TIMESTAMP  WHERE "order".id = NEW.id;  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER set\_order\_timestamp\_on\_status\_change  AFTER UPDATE  ON "order"  FOR EACH ROW  EXECUTE PROCEDURE set\_order\_timestamp\_on\_status\_change\_func();  CREATE OR REPLACE FUNCTION decrease\_shipment\_product\_amount\_on\_assortment\_decrease\_func()  RETURNS TRIGGER AS  $$  DECLARE  remaining INT := NEW.amount;  shipment\_r RECORD;  BEGIN  IF OLD.amount > NEW.amount THEN  FOR shipment\_r IN  (SELECT \*  FROM shipment  WHERE shipment.store\_id = NEW.store\_id AND shipment.product\_id = NEW.product\_id  ORDER BY expiration\_date ASC)  LOOP  IF remaining > 0 THEN  IF remaining > shipment\_r.product\_amount THEN  UPDATE shipment  SET product\_amount = 0  WHERE shipment.id = shipment\_r.id;  remaining := remaining - shipment\_r.product\_amount;  ELSE  UPDATE shipment  SET product\_amount = product\_amount - remaining  WHERE shipment.id = shipment\_r.id;  remaining := 0;  END IF;  END IF;  END LOOP;  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER decrease\_shipment\_product\_amount\_on\_assortment\_decrease  AFTER UPDATE  ON assortment  FOR EACH ROW  EXECUTE PROCEDURE decrease\_shipment\_product\_amount\_on\_assortment\_decrease\_func();  return\_products\_to\_assortment\_on\_order\_cancel\_func()  RETURNS TRIGGER AS  $$  DECLARE  assembling\_r RECORD;  BEGIN  IF NEW."status" = 'cancelled' AND OLD."status" != 'cancelled' THEN  FOR assembling\_r IN  (SELECT \*  FROM assembling  WHERE assembling.order\_id = NEW.id)  LOOP  UPDATE assortment  SET amount = amount + assembling\_r.product\_amount  WHERE assortment.store\_id = NEW.store\_id AND assortment.product\_id = assembling\_r.product\_id;  END LOOP;  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER return\_products\_to\_assortment\_on\_order\_cancel  AFTER UPDATE  ON "order"  FOR EACH ROW  EXECUTE PROCEDURE return\_products\_to\_assortment\_on\_order\_cancel\_func();  CREATE OR REPLACE FUNCTION upd\_price\_on\_assembling\_func()  RETURNS TRIGGER AS  $$  DECLARE  old\_price INT;  new\_price INT;  BEGIN  old\_price := (SELECT total\_price FROM "order" WHERE "order".id = NEW.order\_id);  UPDATE "order"  SET total\_price = total\_price + (SELECT product.price FROM product WHERE product.id = NEW.product\_id)  WHERE "order".id = NEW.order\_id;  new\_price := (SELECT total\_price FROM "order" WHERE "order".id = NEW.order\_id);  IF new\_price >= 1000 AND old\_price < 1000 THEN  UPDATE "order"  SET total\_price = total\_price - 100  WHERE "order".id = NEW.order\_id;  END IF;  RETURN NEW;  END;  $$  LANGUAGE 'plpgsql';  CREATE OR REPLACE TRIGGER upd\_price\_on\_assembling  AFTER INSERT  ON assembling  FOR EACH ROW  EXECUTE PROCEDURE upd\_price\_on\_assembling\_func(); |

Пояснение семантики каждого триггера представлено в списке ниже:

- upd\_price\_on\_assembling обновляет итоговую цену заказа на основе добавляемых продуктов;

- return\_products\_to\_assortment\_on\_order\_cancel возвращает продукты в ассортимент из отменных заказов;

- decrease\_shipment\_product\_amount\_on\_assortment\_decrease при уменьшении количества товара в ассортименте уменьшает остаток в соответствующих поставках. Необходим для корректного учета поставок;

- set\_order\_timestamp\_on\_status\_change\_func обновлении статуса заказа проставляет соответствующую временную метку;

- set\_shipment\_run\_out\_on\_zero\_amount обновляет статус поставки при нулевом остатке;

- decrease\_product\_amount\_on\_new\_assembling уменьшает остаток продукта в магазине при его добавлении в заказ;

- update\_assortment\_on\_shipment\_accepted увеличивает остаток в ассортименте при окончании разгрузки поставки;

- set\_order\_assembled\_on\_assembly\_ready если заказ собран, то меняет ему статус.

Наконец необходимы запланированные функции, которые также обеспечивают корректную работу базы данных. Поскольку используемая база данных не имеет функциональности, позволяющей задать расписание вызовов функций, используя только инструментарий сервера базы данных, предполагается некоторый внешний источник, который будет осуществлять их вызов по расписанию. Такие функции имеет смысл заранее определить в рамках базы данных для их последующей оптимизации движком базы данных. Листинг 5 представляет скрипт для создания функций.

Листинг 5 – Создание функций

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_and\_mark\_expired\_shipments()  RETURNS void  LANGUAGE 'plpgsql' AS  $$  DECLARE  shipment\_r RECORD;  BEGIN  FOR shipment\_r IN  (SELECT \*  FROM shipment  WHERE shipment.expiration\_date < CURRENT\_TIMESTAMP AND shipment.status = 'accepted')  LOOP  UPDATE shipment  SET status = 'expired'  WHERE shipment.id = shipment\_r.id;  UPDATE assortment  SET amount = amount - shipment\_r.product\_amount  WHERE assortment.store\_id = shipment\_r.store\_id AND assortment.product\_id = shipment\_r.product\_id;  END LOOP;  END;  $$;  CREATE OR REPLACE FUNCTION check\_ready\_to\_delivery\_orders\_and\_assign\_couriers()  RETURNS void  LANGUAGE 'plpgsql' AS  $$  DECLARE  order\_r RECORD;  courier\_id\_r INT;  BEGIN  FOR order\_r IN  (SELECT \*  FROM "order"  WHERE "order"."status" = 'assembled')  LOOP  courier\_id\_r := (  SELECT employee\_shift.employee\_id  FROM employee\_shift  WHERE employee\_shift."role" = 'courier' AND employee\_shift.store\_id = order\_r.store\_id  LIMIT 1  );  UPDATE "order"  SET "status" = 'delivering'  WHERE "order".id = order\_r.id;  INSERT INTO delivery (order\_id, courier\_id, "status")  VALUES (order\_r.id, courier\_id\_r, 'scheduled');  END LOOP;  END;  $$; |

Семантика функций описывается в списке ниже:

- check\_ready\_to\_delivery\_orders\_and\_assign\_couriers назначает курьеров, находящихся на смене, для собранных заказов;

- check\_and\_mark\_expired\_shipments помечает просроченные поставки и убирает просроченные продукты из ассортимента.

Наконец необходимо определить представления и роли для повышения удобства работы с базой данных. Представления и роли создаются в листинге 6.

Листинг 6 – Создание ролей и представлений

|  |
| --- |
| CREATE ROLE dba;  GRANT ALL PRIVILEGES ON DATABASE postgres TO dba;  CREATE ROLE manager;  GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO manager;  GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO manager;  CREATE ROLE "user";  GRANT SELECT, UPDATE ON TABLE "order", client TO "user";  GRANT SELECT ON TABLE product, assortment, store, client\_order, client\_assortment TO "user";  CREATE ROLE courier;  GRANT UPDATE ON TABLE delivery TO courier;  GRANT SELECT ON TABLE courier\_delivery TO courier;  CREATE ROLE assembler;  GRANT SELECT, UPDATE ON TABLE assembling, shipment TO assembler;  GRANT SELECT ON TABLE assembler\_assembling TO assembler;  CREATE OR REPLACE VIEW client\_order AS (  WITH order\_product AS (  SELECT assembling.order\_id AS order\_id, product."name" AS product\_name, assembling.product\_amount AS product\_amount  FROM assembling  JOIN product ON product.id = assembling.product\_id  )  SELECT "order".id, "order"."status", "order".client\_id, "order".total\_price, "order".address, "order".creation\_date, "order".close\_date, array\_agg(order\_product.product\_name) as products, array\_agg(order\_product.product\_amount) as amounts  FROM "order"  JOIN order\_product ON order\_product.order\_id = "order".id  GROUP BY "order".id  );  CREATE OR REPLACE VIEW client\_assortment AS (  SELECT store.address, product.name, assortment.amount  FROM assortment  JOIN product ON product.id = assortment.product\_id  JOIN store ON store.id = assortment.store\_id  );  CREATE OR REPLACE VIEW courier\_delivery AS (  SELECT "order".id, "order"."address", employee.id as courier\_id, employee.name, employee.lastname  FROM delivery, "order", employee  WHERE delivery.order\_id = "order".id AND delivery.courier\_id = employee.id AND delivery."status" != 'closed'  );  CREATE OR REPLACE VIEW assembler\_assembling AS (  SELECT "order".store\_id, assembling.order\_id, assembling.product\_amount, product\_location."description"  FROM "order", assembling, product, product\_location  WHERE  "order".id = assembling.order\_id AND  assembling.product\_id = product.id AND  product\_location.product\_id = product.id AND  product\_location.store\_id = "order".store\_id  );  CREATE OR REPLACE VIEW employee\_shift AS (  SELECT employee.id AS employee\_id, employee."role", shift.begin\_date, shift.end\_date, shift.store\_id  FROM employee, shift  WHERE shift.employee\_id = employee.id AND shift.end\_date > CURRENT\_TIMESTAMP  ); |

Для повышения безопасности создаются роли и выдаются соответствующие разрешения:

- dba – администратор базы данных;

- manager – выполняет административную работу в магазине;

- user – клиент предприятия;

- assembler – сборщик заказов;

- courier – доставщик заказов.

Для повышения удобства пользования базой данных созданы представления:

- client\_order агрегирует данные заказов в удобный вид с указанием списка продуктов;

- client\_assortment агрегирует данные по ассортименту в разных магазинах в удобный вид;

- courier\_delivery собирает для курьеров всю необходимую инфомрацию о доставках;

- assembler\_assembling собирает всю необходимую информацию для сборщиков;

- employee\_shift отображает сотрудников, находящихся на смене в данный момент.

## **2.2 Заполнение базы данных**

Для заполнения базы данных был написан скрипт на языке Python с использованием стандартной библиотеки языка, а также сторонних библиотек psycogp2, faker. Полученные данные согласованы, соответствуют предметной области и предполагаемым типам данных. Начало скрипта показано в листинге 6. Полный листинг находится в Приложении А.

Листинг 6 – Первые 90 строчек скрипта заполнения базы данных

|  |
| --- |
| #!/bin/pypy  from faker import Faker  try:  import psycopg2 as ps  from psycopg2.extras import execute\_batch  except:  import psycopg2cffi as ps  from psycopg2cffi.extras import execute\_batch  from itertools import groupby  import random  import json  from datetime import datetime, timezone, timedelta  import sys  CLIENT\_AMOUNT = 150  PRODUCT\_AMOUNT = 931  STORE\_AMOUNT = 500  ORDER\_AMOUNT = 1\_000\_000  OWNER\_IDS = range(STORE\_AMOUNT \* 0 + 1, STORE\_AMOUNT \* 1 + 1)  MANAGER\_IDS = range(STORE\_AMOUNT \* 1 + 1, STORE\_AMOUNT \* 2 + 1)  COURIER\_IDS = range(STORE\_AMOUNT \* 2 + 1, STORE\_AMOUNT \* 3 + 1)  ASSEMBER\_IDS = range(STORE\_AMOUNT \* 3 + 1, STORE\_AMOUNT \* 4 + 1)  random.seed(40)  Faker.seed(42)  fake = Faker(locale="ru\_RU")  conn = ps.connect(  dbname="postgres",  user="postgres",  password="12345678",  host="localhost",  port="5433",  )  cur = conn.cursor()  def ignore\_on\_fail(f):  def g(\*args):  try:  return f(\*args)  except Exception as e:  print(f"continue on err: {e}", file=sys.stderr)  return g  @ignore\_on\_fail  def clients():  data = []  for \_ in range(CLIENT\_AMOUNT):  name = fake.first\_name()  lastname = fake.last\_name()  saved\_addresses = [fake.address() for \_ in range(fake.random\_int(0, 5))]  phone\_number = fake.phone\_number()  email = fake.email()  data.append((name, lastname, saved\_addresses, phone\_number, email))  execute\_batch(  cur,  """  INSERT INTO client (\"name\", lastname, saved\_addresses, phone\_number, email)  VALUES (%s, %s, %s, %s, %s)  """,  data,  )  print(f"clients records {len(data)}")  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/insert/clients.json", "w"  ) as f:  json.dump(data, f)  return data  @ignore\_on\_fail  def products():  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/data/product.json", "r"  ) as f:  data = json.load(f)  data = [(p["name"], p["name"], p["caterogy"], p["img"], p["price"]) for p in data]  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO product ("name", "description", category, image\_url, price)  VALUES (%s, %s, %s::PRODUCT\_CATEGORY\_ENUM, %s, %s)  """,  data,  )  print(f"product records {len(data)}")  return data |

Для проверки корректности заполнения таблиц необходимо выполнить SELECT-запросы к каждой таблице. Результаты обращений представлены на рисунках 3-13.

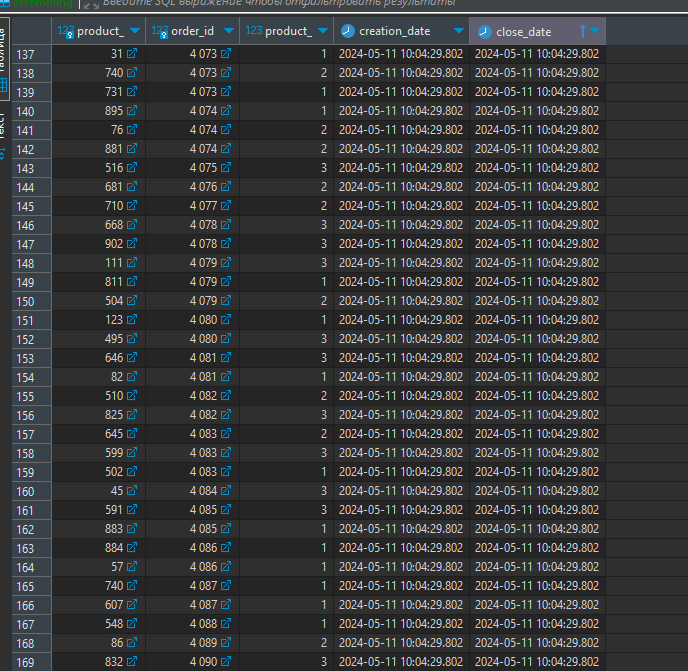


Рисунок 3 – Результат SELECT-запроса к таблице “assembling”

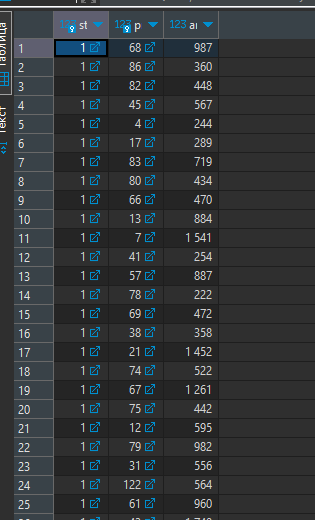


Рисунок 4 – Результат SELECT-запроса к таблице “assortment”

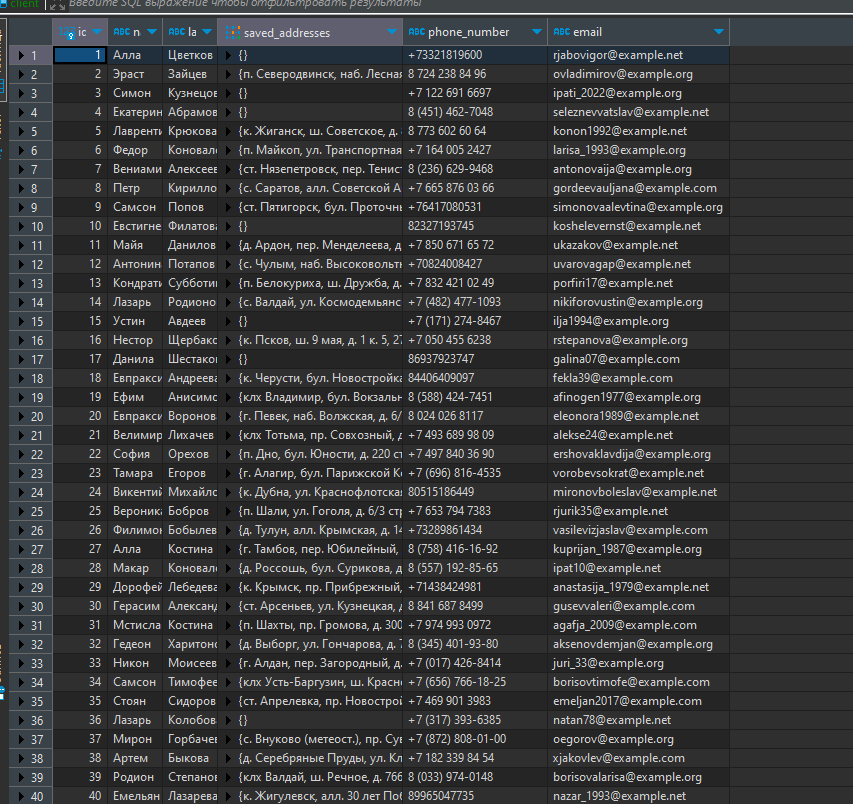


Рисунок 5 – Результат SELECT-запроса к таблице “client”

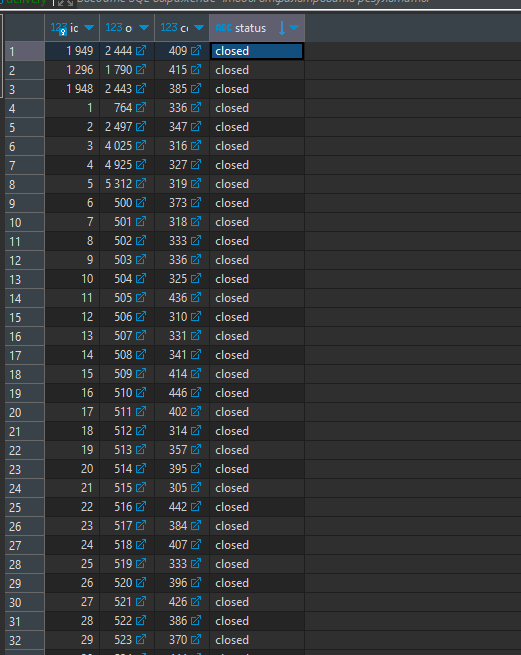


Рисунок 6 – Результат SELECT-запроса к таблице “delivery”

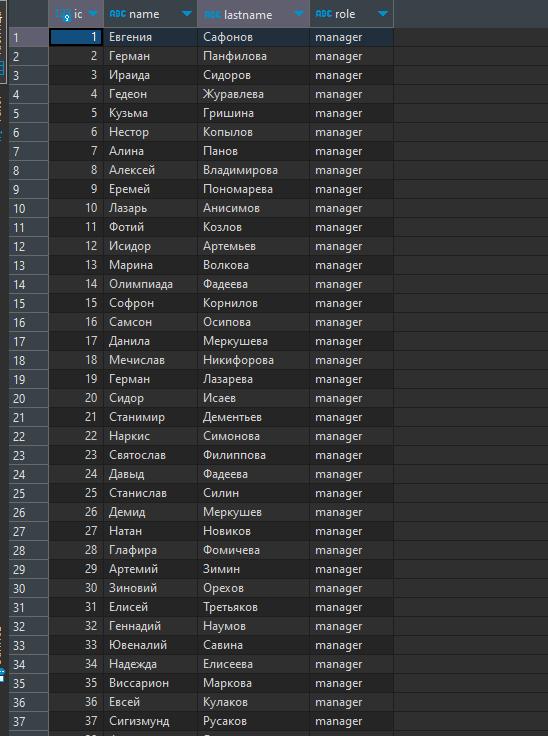


Рисунок 7 – Результат SELECT-запроса к таблице “employee”

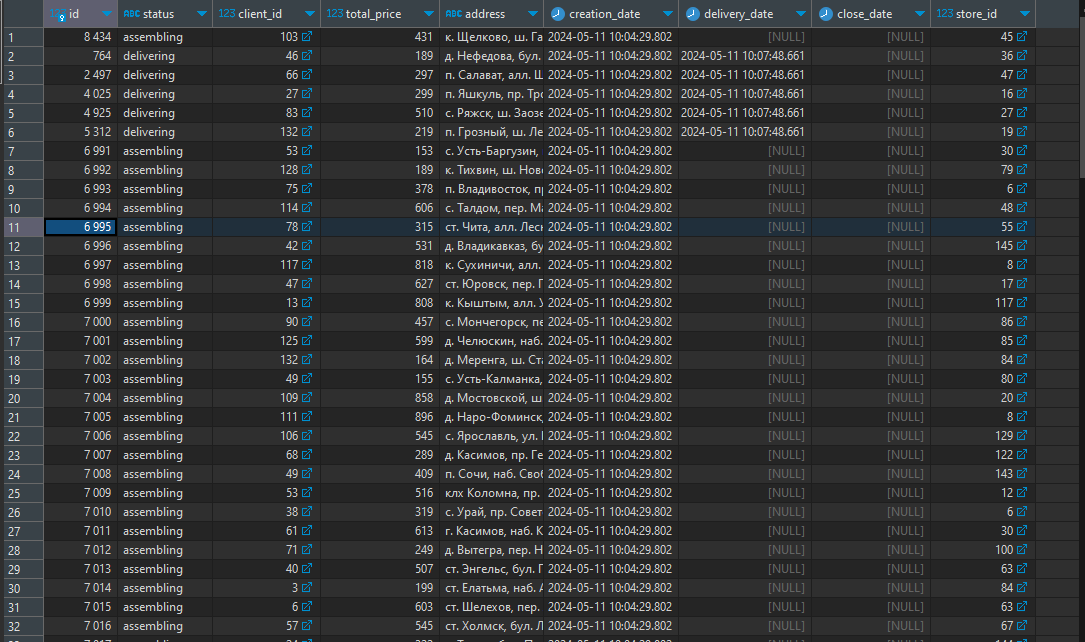


Рисунок 8 – Результат SELECT-запроса к таблице “order”

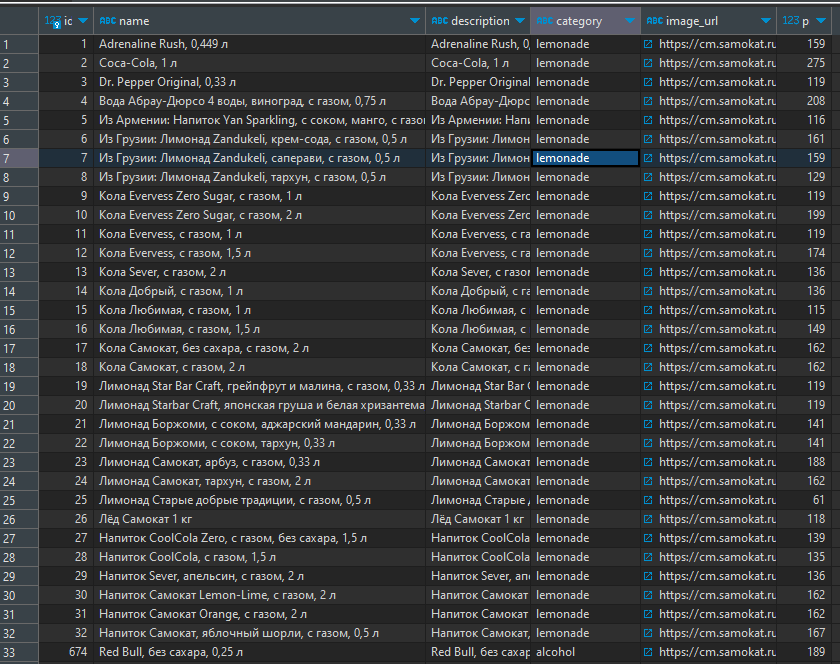


Рисунок 9 – Результат SELECT-запроса к таблице “product”

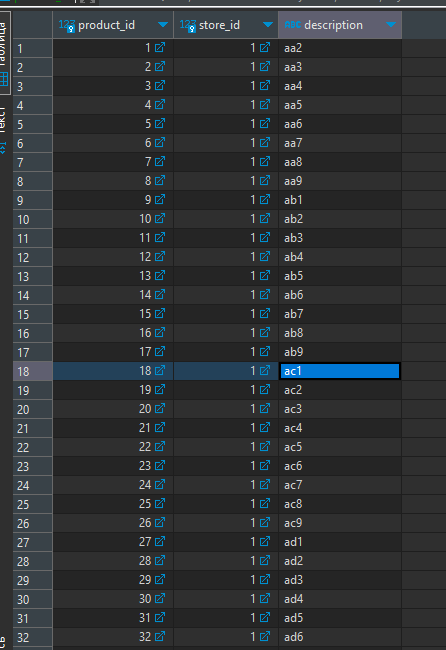


Рисунок 10 – Результат SELECT-запроса к таблице “product\_location”

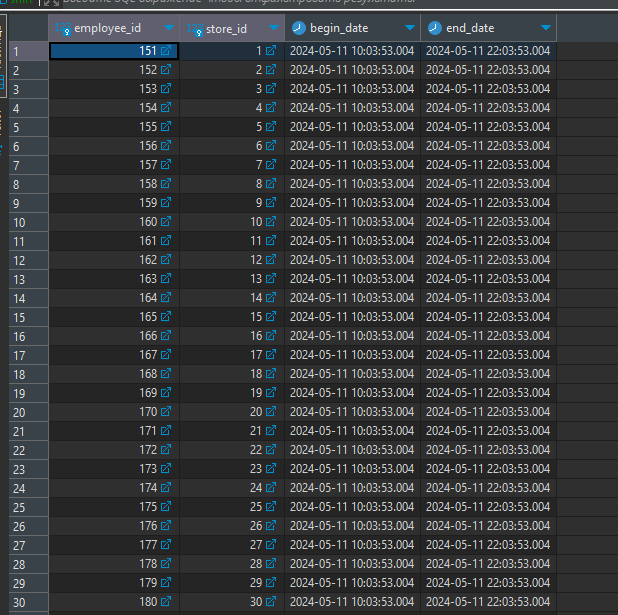


Рисунок 11 – Результат SELECT-запроса к таблице “shift”

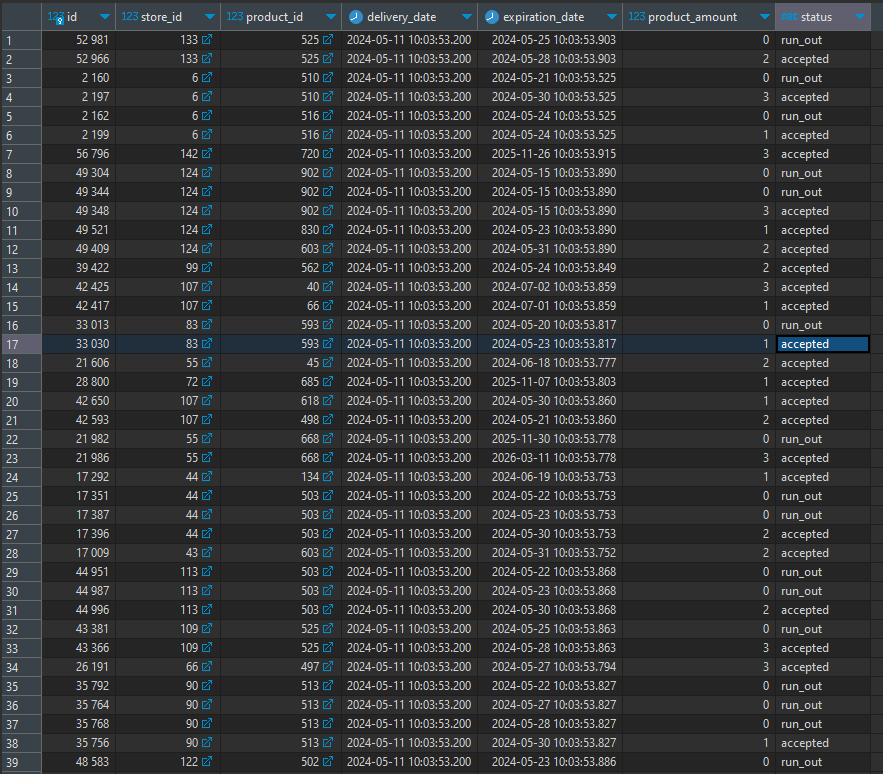


Рисунок 12 – Результат SELECT-запроса к таблице “shipment”

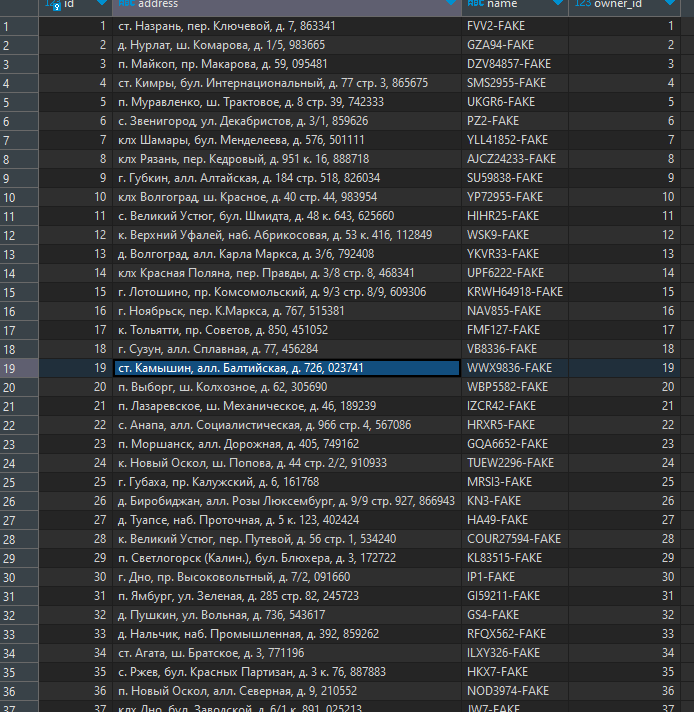


Рисунок 13 – Результат SELECT-запроса к таблице “store”

## **2.3 Сложные запросы**

Для представления данных в удобном для анализа виде написан ряд SQL-запросов. В листингах 7-13 и на рисунках 14-20 представлены соответствующие запросы и результаты их работы.

Листинг 7 – Заказы с списком продуктов и количеством продуктов

|  |
| --- |
| WITH order\_product AS (  SELECT assembling.order\_id AS order\_id, product."name" AS product\_name, assembling.product\_amount AS product\_amount  FROM assembling  JOIN product ON product.id = assembling.product\_id  )  SELECT "order".id, "order"."status", "order".client\_id, "order".total\_price, "order".address, "order".creation\_date, "order".close\_date, array\_agg(order\_product.product\_name) as products, array\_agg(order\_product.product\_amount) as amounts  FROM "order"  JOIN order\_product ON order\_product.order\_id = "order".id  GROUP BY "order".id; |

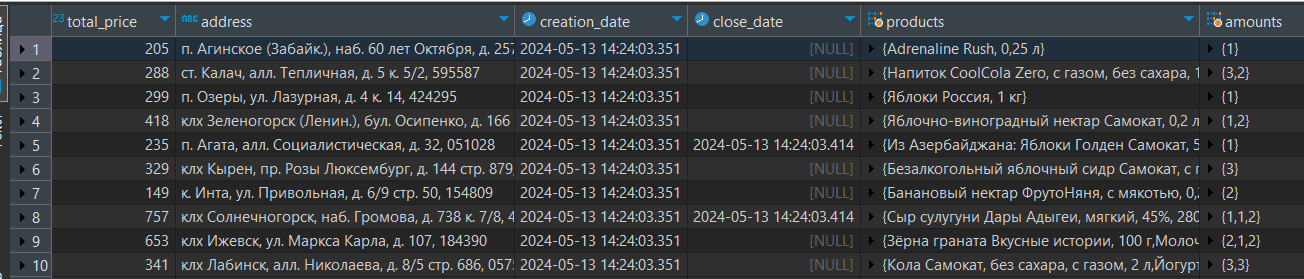


Рисунок 14 – Результат работы первого запроса

Листинг 8 – Списки сотрудников, находящихся на смене в магазинах

|  |
| --- |
| WITH working\_employee AS(  SELECT employee.id AS employee\_id, employee."name", employee.lastname, employee."role", shift.store\_id  FROM employee, shift  WHERE shift.employee\_id = employee.id AND shift.end\_date > CURRENT\_TIMESTAMP  )  SELECT store\_id,  array\_agg(("name", lastname)) filter(WHERE working\_employee."role" = 'manager') as managers,  array\_agg(("name", lastname)) filter(WHERE working\_employee."role" = 'courier') as couriers,  array\_agg(("name", lastname)) filter(WHERE working\_employee."role" = 'assembler') as assemblers  FROM working\_employee  group by store\_id; |

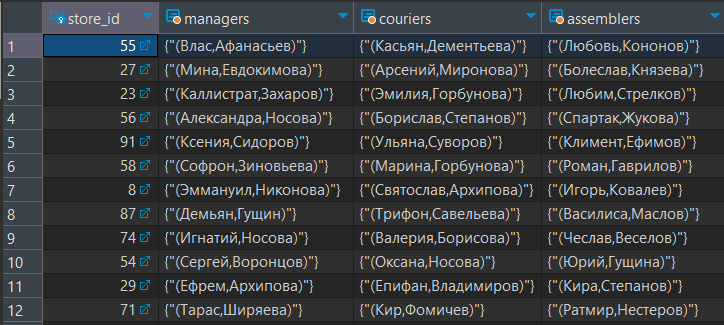


Рисунок 15 – Результат работы второго запроса

Листинг 9 – Магазины с самыми маленькими остатками по продукту и средний остаток в магазине, используется представление client\_assortment.

|  |
| --- |
| select \*, avg(ca.amount) over (partition by ca.address) as avg\_amount  from client\_assortment ca  order by amount asc; |

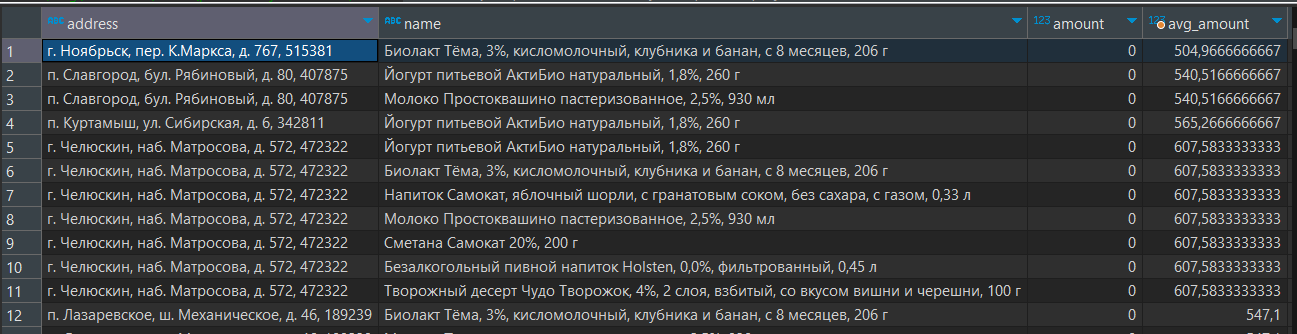


Рисунок 16 – Результат работы третьего запроса

Листинг 10 – Самые продаваемые продукты

|  |
| --- |
| with sales as (select p."name", count(\*) as sales  from product p, assembling a  where p.id = a.product\_id  group by p."name")  select \*  from sales  order by sales.sales desc |

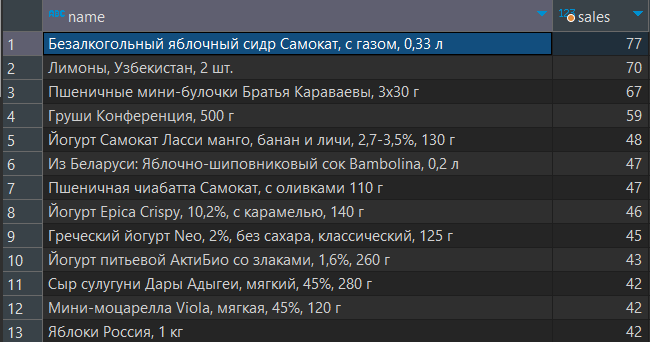


Рисунок 17 – Результат работы четвертого запроса

Листинг 11 – Среднее время сборки по магазинам

|  |
| --- |
| select s."name", avg(a.close\_date - a.creation\_date)  from store s, assembling a, "order" o  where a.order\_id = o.id and s.id = o.store\_id and a.close\_date is not null  group by s."name" |

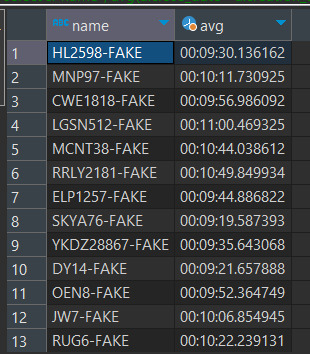


Рисунок 18 – Результат работы пятого запроса

Листинг 12 – Процент списаний партий по магазинам

|  |
| --- |
| select s2."name", 1.0 \* (count(\*) filter(where s.status = 'expired')) / (count(\*) filter(where s.status != 'expired')) as perc  from shipment s, store s2  where s2.id = s.store\_id  group by s2."name" |

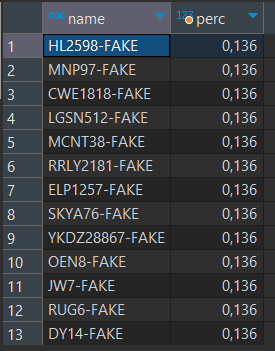


Рисунок 19 – Результат работы шестого запроса

Листинг 13 – Средняя цена товаров по категориям, которые были доставлены в магазин с определенным названием (в листинге - ‘FVV2-FAKE’)

|  |
| --- |
| select p.category , avg(p.price) as avg\_price  from product p  join shipment s on s.product\_id = p.id  join store s2 on s2.id = s.store\_id  where s2.name = 'FVV2-FAKE'  group by p.category |

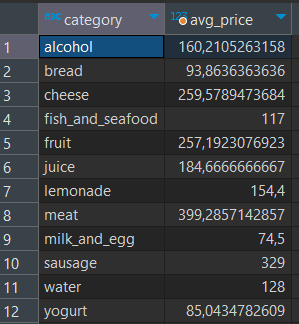


Рисунок 20 – Результат работы седьмого запроса

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной работы выполнено следующее:

- Проведен анализ предметной области, выделены основные сущности и процессы;

- Спроектирована база данных, разработаны инфологическая и даталогическая модель базы данных;

- Написан скрипт создания таблиц базы данных и триггеров, обеспечивающих согласованность данных, представлений, функций, ролей, индексов, внешних ключей;

- Разработаны и написаны сложные SQL-запросы;

В результате курсовой работы, была реализована система, позволяющая выполнять заказы продуктов.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1) Скворцова М.А. Методические указания по выполнению курсовой работы по курсу «Базы данных»

2) Иванова Г.С., Ничушкина Т.Н. Оформление текстовых документов. Методические указания по оформлению расчетно-пояснительных записок дипломных и квалификационных работ. -М.:Издательство МГТУ им. Н.Э. Баумана, 2004. -10с.

3) [Документация библиотеки psycopg] URL: <https://www.psycopg.org/>

4) [Документация библиотеки faker] URL: <https://faker.readthedocs.io/en/master/>

# **ПРИЛОЖЕНИЕ А**

Листинг А.1 – Программа для генерации данных

|  |
| --- |
| #!/bin/pypy  from faker import Faker  try:  import psycopg2 as ps  from psycopg2.extras import execute\_batch  except:  import psycopg2cffi as ps  from psycopg2cffi.extras import execute\_batch  from itertools import groupby  import random  import json  from datetime import datetime, timezone, timedelta  import sys  CLIENT\_AMOUNT = 150  PRODUCT\_AMOUNT = 931  STORE\_AMOUNT = 500  ORDER\_AMOUNT = 1\_000\_000  OWNER\_IDS = range(STORE\_AMOUNT \* 0 + 1, STORE\_AMOUNT \* 1 + 1)  MANAGER\_IDS = range(STORE\_AMOUNT \* 1 + 1, STORE\_AMOUNT \* 2 + 1)  COURIER\_IDS = range(STORE\_AMOUNT \* 2 + 1, STORE\_AMOUNT \* 3 + 1)  ASSEMBER\_IDS = range(STORE\_AMOUNT \* 3 + 1, STORE\_AMOUNT \* 4 + 1)  random.seed(40)  Faker.seed(42)  fake = Faker(locale="ru\_RU")  conn = ps.connect(  dbname="postgres",  user="postgres",  password="12345678",  host="localhost",  port="5434",  )  cur = conn.cursor()  def ignore\_on\_fail(f):  def g(\*args):  try:  return f(\*args)  except Exception as e:  print(f"continue on err: {e}", file=sys.stderr)  return g  @ignore\_on\_fail  def clients():  data = []  for \_ in range(CLIENT\_AMOUNT):  name = fake.first\_name()  lastname = fake.last\_name()  saved\_addresses = [fake.address() for \_ in range(fake.random\_int(0, 5))]  phone\_number = fake.phone\_number()  email = fake.email()  data.append((name, lastname, saved\_addresses, phone\_number, email))  execute\_batch(  cur,  """  INSERT INTO client (\"name\", lastname, saved\_addresses, phone\_number, email)  VALUES (%s, %s, %s, %s, %s)  """,  data,  )  print(f"clients records {len(data)}")  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/clients.json", "w"  ) as f:  json.dump(data, f)  return data  @ignore\_on\_fail  def products():  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/data/product.json", "r"  ) as f:  data = json.load(f)  data = [(p["name"], p["name"], p["caterogy"], p["img"], p["price"]) for p in data]  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO product ("name", "description", category, image\_url, price)  VALUES (%s, %s, %s::PRODUCT\_CATEGORY\_ENUM, %s, %s)  """,  data,  )  print(f"product records {len(data)}")  return data  @ignore\_on\_fail  def stores():  data = []  for i in range(1, STORE\_AMOUNT + 1):  address = fake.address()  owner\_id = i  name = fake.nic\_handle()  data.append((address, name, owner\_id))  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO store ("address", "name", "owner\_id")  VALUES (%s, %s, %s)  """,  data,  )  print(f"store records {len(data)}")  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/stores.json", "w"  ) as f:  json.dump(data, f)  return data  @ignore\_on\_fail  def employees():  data = []  for \_ in OWNER\_IDS: # owner  name = fake.first\_name()  lastname = fake.last\_name()  role = "manager"  data.append((name, lastname, role))  for \_ in OWNER\_IDS: # managers  name = fake.first\_name()  lastname = fake.last\_name()  role = "manager"  data.append((name, lastname, role))  for \_ in COURIER\_IDS: # couriers  name = fake.first\_name()  lastname = fake.last\_name()  role = "courier"  data.append((name, lastname, role))  for \_ in ASSEMBER\_IDS: # assemblers  name = fake.first\_name()  lastname = fake.last\_name()  role = "assembler"  data.append((name, lastname, role))  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO employee ("name", lastname, "role")  VALUES (%s, %s, %s::EMPLOYEE\_ROLE\_ENUM)  """,  data,  )  print(f"employee records {len(data)}")  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/employees.json", "w"  ) as f:  json.dump(data, f)  return data  @ignore\_on\_fail  def product\_locations():  data = []  descs = [  "".join(chr(x) + chr(y) + chr(z))  for x in range(ord("a"), ord("z") + 1)  for y in range(ord("a"), ord("z") + 1)  for z in range(ord("1"), ord("9") + 1)  ]  for store\_id in range(1, STORE\_AMOUNT + 1):  for product\_id in range(1, PRODUCT\_AMOUNT + 1):  desc = descs[product\_id]  data.append((product\_id, store\_id, desc))  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/product\_locations.json",  "w",  ) as f:  json.dump(data, f)  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO product\_location (product\_id, store\_id, "description")  VALUES (%s, %s, %s)  """,  data,  )  print(f"product\_location records {len(data)}")  return data  @ignore\_on\_fail  def shifts():  store\_id = lambda x: (x - 1) % STORE\_AMOUNT + 1  data = []  start = datetime.now(timezone(timedelta(hours=3)))  end = start + timedelta(hours=12)  for employee\_id in MANAGER\_IDS:  data.append((employee\_id, store\_id(employee\_id), start, end))  for employee\_id in COURIER\_IDS:  data.append((employee\_id, store\_id(employee\_id), start, end))  for employee\_id in ASSEMBER\_IDS:  data.append((employee\_id, store\_id(employee\_id), start, end))  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/shifts.json", "w"  ) as f:  json.dump(data, f, default=str)  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO shift (employee\_id, store\_id, begin\_date, end\_date)  VALUES (%s, %s, %s, %s)  """,  data,  )  print(f"shift records {len(data)}")  return data  @ignore\_on\_fail  def shipments():  data = []  exp\_data = []  exp\_shipments = [  (random.randint(1, 931), -1), # expired  (random.randint(1, 931), -1), # expired  ]  shipments = [  \*[  [random.randint(1, 90), random.randint(30, 60)] for \_ in range(50)  ], # lemonade  \*[  [random.randint(91, 167), random.randint(20, 40)] for \_ in range(50)  ], # cheese  \*[[random.randint(890, 931), random.randint(1, 4)] for \_ in range(50)], # bread  \*[  [random.randint(493, 536), random.randint(10, 20)] for \_ in range(50)  ], # meat  \*[  [random.randint(537, 663), random.randint(5, 20)] for \_ in range(50)  ], # yogurt  \*[  [random.randint(810, 889), random.randint(10, 30)] for \_ in range(50)  ], # fruit  \*[  [random.randint(810, 889), random.randint(10, 30)] for \_ in range(50)  ], # fruit  \*[  [random.randint(664, 743), random.randint(360, 720)] for \_ in range(50)  ], # alcohol  ]  for store\_id in range(1, STORE\_AMOUNT + 1):  for product\_id, exp in shipments:  product\_amount = random.randint(200, 1000)  expiration\_date = datetime.now(timezone(timedelta(hours=-3))) + timedelta(  days=exp  )  data.append([store\_id, product\_id, expiration\_date, product\_amount])  for product\_id, exp in exp\_shipments:  product\_amount = random.randint(10, 100)  expiration\_date = datetime.now(timezone(timedelta(hours=-3))) + timedelta(  days=exp  )  exp\_data.append((store\_id, product\_id, expiration\_date, product\_amount))  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/shipments.json", "w"  ) as f:  json.dump(data, f, default=str)  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO shipment (store\_id, product\_id, expiration\_date, product\_amount)  VALUES (%s, %s, %s, %s)  """,  data,  )  execute\_batch(  cur,  """  INSERT INTO shipment (store\_id, product\_id, expiration\_date, product\_amount)  VALUES (%s, %s, %s, %s)  """,  exp\_data,  )  cur.execute(  """  UPDATE shipment  SET "status" = 'accepted'  WHERE shipment.id < %s  """,  (str(int(len(data) \* 0.95)),),  )  cur.execute(  """  SELECT check\_and\_mark\_expired\_shipments()  """  )  print(f"shipment records {len(data)+len(exp\_data)}")  return list(map(lambda x: [x[0] + 1, \*x[1]], enumerate(data)))  @ignore\_on\_fail  def orders(shipments):  assems = []  orders = []  def dec(x, i, y):  x[i] -= y  return y  for order\_id in range(1, ORDER\_AMOUNT + 1):  client\_id = random.randint(1, CLIENT\_AMOUNT)  address = fake.address()  shs = []  while len(shs) < 2:  store\_id = random.randint(1, STORE\_AMOUNT + 1)  shs = list(filter(lambda x: x[1] == store\_id and x[4] > 5, shipments))  assemblings = [  (p, order\_id, sum(x[2] for x in s))  for p, s in groupby(  (  (shipment[2], order\_id, dec(shipment, 4, random.randint(1, 3)))  for shipment in random.sample(  shs, k=random.randint(1, min(len(shs), 3))  )  ),  key=lambda x: x[0],  )  ]  assems += assemblings  orders.append((client\_id, address, store\_id))  assems = list(set({(x[0], x[1]): x for x in assems}.values()))  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/orders.json", "w"  ) as f:  json.dump(orders, f, default=str)  with open(  "/home/vzalygin/repos/bmstu-ics6/cource-papers/db/~insert/assemblings.json", "w"  ) as f:  json.dump(assems, f, default=str)  print("insert into db")  execute\_batch(  cur,  """  INSERT INTO "order" (client\_id, "address", store\_id)  VALUES (%s, %s, %s)  """,  orders,  )  execute\_batch(  cur,  """  INSERT INTO assembling (product\_id, order\_id, product\_amount)  VALUES (%s, %s, %s)  """,  assems,  )  print("insert into db")  cur.execute(  """  UPDATE assembling  SET close\_date = CURRENT\_TIMESTAMP  WHERE assembling.order\_id in (SELECT assembling.order\_id FROM assembling LIMIT %s)  """,  (str(int(len(assems) \* 0.70)),),  )  print("insert into db")  cur.execute(  """  UPDATE "order"  SET "status" = 'cancelled'  WHERE "order".id < %s  """,  (str(int(ORDER\_AMOUNT \* 0.05)),),  )  print(f"order records {ORDER\_AMOUNT}")  print(f"assembling records {len(assems)}")  @ignore\_on\_fail  def deliveries():  cur.execute(  """  SELECT check\_ready\_to\_delivery\_orders\_and\_assign\_couriers();  """  )  cur.execute(  """  SELECT count(\*) FROM delivery  """  )  (delivery\_amount,) = cur.fetchone()  cur.execute(  """  UPDATE delivery  SET "status" = 'on\_the\_way'  WHERE delivery.id < %s  """,  (str(int(delivery\_amount \* 0.70)),),  )  cur.execute(  """  UPDATE delivery  SET "status" = 'closed'  WHERE delivery.id < %s  """,  (str(int(delivery\_amount \* 0.50)),),  )  print(f"delivery records {delivery\_amount}")  # static  clients()  conn.commit()  products()  conn.commit()  stores()  employees()  conn.commit()  product\_locations()  conn.commit()  # dynamic  shifts()  s = shipments()  conn.commit()  orders(s)  conn.commit()  deliveries()  conn.commit()  print("db filled successfully") |